The DUST procedure for finding low-complexity sequences

DUST finds regions within a given nucleotide sequence that have low complexity and masks them. DUST is a heuristic algorithm that has been frequently used both inside and outside BLAST for many years. DUST computes scores for substrings based on how often different three-letter sequences (referred to below as *triplets*) occur in the substring. More occurrences of a triplet lead to a higher score that grows quadratically with the number of occurrences. Higher scores are considered to imply lower complexity. We now specify more details, including the current parameter settings, in DUST.

The DUST algorithm examines the input sequence using all windows of 64 consecutive bases; the window size of 64 is independent of the size of the genome being masked. At the ends of the sequence, shorter windows are also considered, with the shortest one being 3 bases. Each window is considered independently: the bases masked in a given window do not depend on the bases masked in any other window. For each window, a maximum of one contiguous substring is masked. The full database sequence is masked by taking the union of the bases masked in all windows.

For each window, all groups of three consecutive bases are concatenated to form triplets. DUST assumes that if a region within the window has low complexity, then one or more triplets will occur multiple times in the window. The algorithm uses a running count of triplets in an interval to measure complexity. The running count is defined as follows. Let S be any triplet and I any substring of the sequence being masked, we define

$$\mathrm{count}(S; I) = \text{the number of times } S \text{ occurs in } I.$$

For an index i, let S_i denote the triplet $a_i a_{i+1} a_{i+2}$. Then if $a_j \ldots a_k$ is any substring of the sequence being masked

$$\mathrm{running\_count}(a_j \ldots a_k) = \sum_{i=j}^{k-2} \left(\mathrm{count}(S_i; a_j \ldots a_{i+2}) - 1\right).$$

For example, if triplet S occurs 4 times in $a_j \ldots a_k$, then it contributes $0+1+2+3$ to the running count. DUST assigns a complexity score to the interval I using the formula

$$\mathrm{dust\_score}(I) = \begin{cases} 10 \times \frac{\mathrm{running\_count}(I)}{(\mathrm{len}(I)-3)} & \text{if } \mathrm{len}(I) > 3; \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm may mask one sufficiently high scoring interval in a window using the following algorithm. Let $W = a_\ell \ldots a_m$ be a window starting at character ℓ and extending to character m; W will have length 64 except at the ends of sequences.

1. For $i = \ell + 2, \ldots, m$, calculate the score of each prefix of W of the form $a_\ell \ldots a_i$.
2. Let $a_\ell \ldots a_p$ be the high-scoring prefix found in step 1. If the score of $a_\ell \ldots a_p$ exceeds a threshold (set to 20 by default), find a subsequence $a_j \ldots a_k$ with maximal score. This final subsequence is the region that is masked in the current window. Observe that in step 1, the left endpoint was fixed at ℓ, but for the high-scoring subsequence $\ell \leq j < k \leq p$.

Step 1 searches for a large subsequence within each window that has low complexity (high score). If a subsequence of sufficiently high score is found, then step 2 refines the subsequence into a possibly smaller subsequence with higher score. The end result is the identification of a maximum of one interval within each window that has low complexity. DUST is conservative, in that if a high-complexity region is flanked by two low-complexity regions, whose overall score would exceed the score threshold, then only the worst low-complexity region is masked. This prevents the algorithm from erroneously masking too many bases, but in the process can miss low-complexity regions that are close together.

For the example given in the Introduction, the DUST output is:

```
GGTTGGTcaaataaaaagtgatgtatgaaaaagagg
caaaacaacaagaagaaaagattgaaaaaatgagag
ctgaagatggtgaaaaTTATGACATCAAAAAGCAGG
```

with the masked portion in lower case. Interestingly, the five consecutive As near the end are not masked. This is partly because the preceding `TTATGACATC` contains infrequent triplets which lowers the score sufficiently.

Further development of DUST is ongoing, with the twin goals of improving speed and of improving masking for the database searching application.

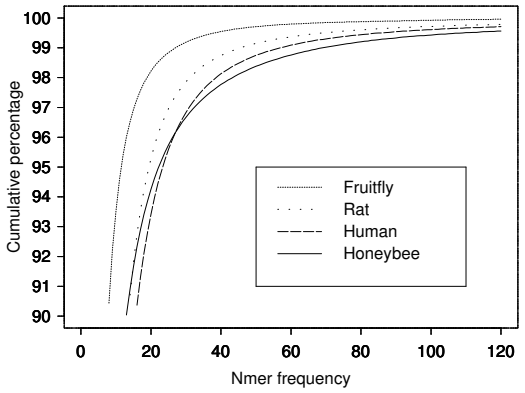

Fig. S1. Portions of cumulative Nmer frequency distributions for human, rat, honeybee, and fruitfly genomes. Distributions for mouse and D. pseudoobscura are not shown as they are indistinguishable from those for human and fruitfly, respectively.

Table S1. WinMask parameters used in WindowMasker tests on six genomes.

Genome	N	$T_{\mathrm{threshold}}$	T_{extend}	T_{high}	T_{low}
Human build 34.1	15	86	57	154	16
Mouse build 32.1	15	74	50	138	15
Mouse build 33.1	15	77	50	141	15
Fruitfly build 6.3	13	39	28	61	8
Honeybee build 1.1	13	110	70	210	13
Pseudoobscura	13	39	28	62	9
Rat build 2.1	15	70	46	127	14

Table S2. Histogram for number of regions of given length masked by one method and not masked at all by the other method.

length (bp)	≤ 100	100 - 1000	1000 - 2500	2500 - 5000	> 5000
Human build 34.1					
RepeatMasker	157165	227445	118	1	0
WindowMasker	6088908	14071	23	7	1
Mouse build 32.1					
RepeatMasker	142200	202417	30	0	0
WindowMasker	5066333	17978	100	11	2
Mouse build 33.1					
RepeatMasker	146588	213224	35	0	0
WindowMasker	4728245	19290	298	36	5
Fruitfly build 6.3					
RepeatMasker	1697	357	9	1	0
WindowMasker	596121	791	18	1	1